MODELING, IDENTIFICATION AND CONTROL, 1983, VOL. 4, NO. 3, 139–165
doi:10.4173/mic.1983.3.2

Partitioning and tearing of networks—applied to process flowsheeting

TRULS GUNDERSEN† and TERJE HERTZBERG‡

Keywords: *Decomposition, graph theory, process flowsheeting, modular systems.*

A review of the methods available for identification of the computational sequence in modular process simulators (partitioning and tearing) is followed by the presentation of a new very efficient and close-to-optimal routine for tearing.

The problem of partitioning can be solved in computer times that are linear functions of the number of unit modules (vertices in the graph). The algorithm of Johns has been found to execute faster than the later and far better known algorithm of Tarjan. These methods are almost identical in idea but use different techniques in the book-keeping.

While partitioning is simple and straightforward, tearing belongs to the class of problems that are stated to be NP-complete (Karp) and thus computationally intractable. No matter how efficient the algorithms are, there will always be an upper limit on the dimension and complexity of the graphs that can be solved in reasonable amounts of computer time. It is the object of the research in this field to push this limit beyond the size of what are considered feasible problems to deal with from a numerical point of view.

Branch and bound methods have proven to overcome the inherent dimensionality problems of algorithms based on enumeration or dynamic programming. Two of the most efficient methods are the Pho/Lapidus and the Johns/Müller algorithms. The latter is employed in Flowpack II (ICI/Linde) and plays a significant role in utilizing the concept of subnetworking.

A new and very simple method for the weighted case of tearing is presented which may serve as an upper bound algorithm in a branch and bound strategy. It is based on repeated application of a partitioning procedure such as Johns' or Tarjan's routine. Between each partitioning, all input streams to a selected vertex are removed from the graph (torn). The algorithm has been run on 10 test problems and executes slightly faster than $0(n^2)$ with a very small constant factor. The method is believed to be faster than the non-optimal algorithm of Kevorkian which is based on graph simplification.

The group of test problems involves the classical graphs from the chemical engineering literature and in addition the very complex graph of a heavy water plant on which tearing by hand is impossible. The new tearing algorithm finds the true minimum number of tears for 6 of the test problems, while the number of tears found for the remaining 4 problems is only *one more* than the optimum value. The decomposition of the heavy water plant with 109 vertices, 163 edges and 13746 elementary cycles is done in less than a second on a UNIVAC-1100/62.

1. Introduction

Although the identification of the computational sequence in modular process simulators has been discussed for more than twenty years, the field is an area of current

Received 6 October 1982.

† Research Centre, Norsk Hydro a.s., Porsgrunn, Norway.

‡ Laboratory of Chemical Engineering, The Norwegian Institute of Technology, The University of Trondheim, Norway.

§ Parts of this paper was presented at the International Symposium on Process Systems Engineering, Kyoto, Japan, August 23–27, 1982 under the title 'Partitioning and Tearing Chemical Process Flowsheets'.

interest and new algorithms are constantly being presented. Since tearing belongs to the class of problems that are stated to be NP-complete, there is a need for even more efficient algorithms in order to handle the increasing size of flowsheeting problems dealt with by chemical engineers.

The need for efficient decomposition routines is even more obvious when using subnetworking than in traditional sequential modular packages. In subnetworking (Berger and Perris 1979) as it is implemented in Flowpack II (ICI/Linde) the unit modules (mixer, flash, reactor etc.) are further broken down into elementary operations like flow adder, temperature from enthalpy, vapour pressure from temperature etc. The number of such elementary blocks may be ten times larger than the number of traditional modules in the flowsheet. Decomposition by hand may then be impossible and Flowpack II employs the branch and bound tearing algorithm of Johns and Müller (1976).

In a recent thesis, Gundersen (1982) has given a review and evaluation of the methods for decomposition of large scale chemical engineering systems stated in the unit modular or the equation based approach. This paper presents some of the results from the unit modular approach together with a brief review of the most important methods that are available for general decomposition.

Graph theory is fundamental in the development of algorithms for partitioning and tearing. The networks or information diagrams from chemical plants are similar to graphs from other disciplines and some of the algorithms are thus applicable not only to chemical engineering problems. It has been attempted in this work to 'look beyond the scientific fences' and find out how similar problems are solved in other fields (operations research, electrical engineering, applied mathematics, computer science etc.). Since graph theory is commonly used for decomposition, definition of the basic terms are not given here, and readers are referred to the text books by Berge (1962) and Harary (1969).

2. Partitioning

The purpose of partitioning is to decompose a problem into subproblems that can be solved in a precedence ordered sequence. In the unit modular approach, these subproblems correspond to strongly connected components of the directed graph (maximal cyclical nets). In the equation based approach, partitioning is achieved by permuting rows and columns of the boolean occurrence matrix in order to obtain a block lower triangular form.

An early and fundamental work on decomposition in chemical engineering was made by Sargent and Westerberg (1964). They presented a list processing technique to trace paths backwards from unit to unit. When a unit is encountered twice, a loop is detected, and the units of this loop are grouped into a single unit. Sargent and Westerberg used separate lists of the inputs to each unit, and a pointer to the next input to be examined made a depth first search possible.

For several years, this method was not fully accepted, and partitioning was done by boolean matrix techniques. It was Norman (1965) who presented the idea of matrix multiplication to locate the cycles of a directed graph. The concept of loop-detection by inspecting the main diagonal of increasing powers of the boolean adjacency matrix is the basis for the algorithm presented by Himmelblau (1966). The strongly connected components are found by the logical intersection of the reachability matrix and its transpose.

Several authors (Harary 1962, Thorelli 1966, Ledet and Himmelblau 1970, Mah 1974) have shown that the reachability matrix can be found efficiently by using the rules of boolean algebra:

$$R = A + A^2 + \ldots + A^n = A(I + A)^{n-1}$$

Some authors define the main diagonal of the reachability matrix to be equal to one (disregard self-loops):

$$R' = R + I$$

Since A^n only can produce new diagonal elements, the new reachability matrix can be obtained by:

$$R' = I + A + \ldots + A^{n-1} = (I + A)^{n-1}$$

Instead of computing each power of $A + I$ to obtain R', one should multiply the matrix by itself to obtain the second, the fourth, the eighth etc. powers until the power is equal to or greater than $n - 1$.

The digraph corresponding to the reachability matrix is in graph theory referred to as the transitive closure. According to Bowie (1976), who has given an excellent review of the applications of graph theory in computer science, one of the first and most commonly used algorithms for finding the transitive closure is due to Warshall (1962). The algorithm takes on the order of n^3 operations and works briefly as follows. Each column of the adjacency matrix is scanned in turn. If a non-zero is found in row i, when scanning column k, row i is replaced by the union (logical OR) of row i and row k. The reachability matrix is computed in one pass over the adjacency matrix.

At the same time and independently, Baker (1962) presented a method quite similar to Warshall's method. Since Baker, however, scanned rows of the adjacency matrix instead of columns, several passes must be made until a complete pass is made which does not change the matrix.

Thorelli (1966) has presented an algorithm which is said to be "especially suited for cases (graphs) where the number of edges is relatively small compared to the number of vertices". The graphs from chemical engineering problems *are* most often sparse, but since they are connected, the number of edges is slightly *greater* than the number of vertices, and the number of non-zero elements in the reachability matrix is at least $n^2/2$.

In the chemical engineering literature, Mah (1974) has presented an algorithm based on arc augmentation. Gundersen (1982) has shown, however, that this algorithm is identical to Warshall's algorithm.

Purdom (1970) and Munro (1971) both compute the strongly connected components as an intermediate step in obtaining the transitive closure. Thus it is not necessary for decomposition purposes to find the reachability matrix, and all methods for finding the maximal cyclical nets based on the logical intersection of the reachability matrix and its transpose are inefficient.

Hlaváček (1977) has among others argued that the path tracing methods are more complicated to program than the adjacency matrix based methods. When using computer languages with data structures for list processing like PASCAL (Jensen and Wirth 1978) and SIMULA 67 (Dahl, Myrhaug and Nygaard 1968) this argument is not valid. More important, however, is the fact that the path tracing methods which was first presented by Sargent and Westerberg use fewer computation steps than the boolean matrix related methods.

The best known method for partitioning is the work of Tarjan (1972). The algorithm is based on a depth-first search, which visits the vertices of a graph in a forward direction as long as possible. The major difference between the work of Tarjan and those of Sargent and Westerberg, Munro, and Purdom is that in the latter ones much work is done in the relabelling procedure when lists are merged (vertices are grouped into supernodes). Having made a careful study of what a depth-first search actually does to a directed graph, Tarjan has presented an algorithm which requires no merging. The algorithm is very economic in the sense that each edge is explored just once, and the merging of vertices (with a corresponding merging of adjacency lists) is avoided.

While the well-known algorithm of Tarjan has been adopted in several scientific disciplines, a much less known and earlier method has been presented by Johns (1970). These methods are almost identical in idea, share the characteristic that each edge is explored only once and execute in computer times that are linear functions of the number of vertices in the graph. The only significant difference is in the book-keeping of locating the beginning of each full partition. This difference, however, is only a question of technique, not of logic.

Path tracing methods have also been presented by Steward (1965), Billingsley (1967), Christensen and Rudd (1969) and Forder and Hutchison (1969). Some authors have tried to combine the path tracing approach with the boolean matrix technique by taking the best features of each of these approaches. The best known of these combined algorithms are the works of Jain and Eakman (1971) and Kehat and Shacham (1973). Both methods use a compact storage of the adjacency matrix to avoid both the large storage requirements and the unecessary operation on zeros.

Finally in this section on partitioning, a brief review of the methods for enumerating the elementary cycles of a directed graph is given. The elementary cycles are required by those tearing methods that are based on covering the cycle matrix.

Jain and Eakman (1971) used a complete tree structuring technique to each maximal cyclical net. By rotating the starting node (root) all combinations are enumerated. Since the elementary cycles may be detected more than once, Jain and Eakman assigned a unique integer code to each cycle in order to avoid the duplication of cycles.

It was Tiernan (1970) who solved the problem that the elementary cycles were detected more than once during the search. The solution is ingenious but simple and is based on the condition that in the elementary path building, each vertex must have a value (number) which is greater than the starting vertex of the path. The elementary paths are being formed in a depth first manner, and vertex closure assures that no elementary paths are considered more than once. Weinblatt (1972) has presented an alternative algorithm which minimizes the number of examinations of each individual edge, but only by substantial book-keeping efforts.

Tarjan (1973) has presented two examples where the algorithms of Tiernan and Weinblatt would lead to exhaustive search and prohibitive computer times. In order to overcome these combinatorial problems, Tarjan uses Tiernan's backtracking technique restricted so that only fruitful paths are explored. A marking procedure is the key to avoiding the unnecessary searches which may occur when using Weinblatt's algorithm or the original method of Tiernan.

Tsukiyama *et al.* (1975) claim to have developed an algorithm which is substantially faster than Tarjan's method. One of their arguments is that Tarjan takes no account of the fact that any cycle of a directed graph is contained in a strongly connected component. This is not true, since Tarjan has presented *two* algorithms, one for detecting the strongly connected components and another for finding the elementary cycles. It is obvious that these algorithms should be applied sequentially, thus the elementary cycle algorithm would operate on one maximal cyclical net at a time.

Equation based approach

In the equation based approach, partitioning may be achieved by the same technique as in the modular approach. Before a partitioning procedure like Tarjan's or Johns' method is applied, however, one has to sort out the disjoint subsets (if any), and define the flow of information by assigning an output set.

When it comes to tearing, however, it is not clear in the equation based approach, whether tearing reduces the overall computation time for example compared with sparse techniques for solving each subsystem of equations (partitions) simultaneously. Reliability of the resulting numerical scheme should also be taken into account, but these topics are not discussed here. All that we say is that the graph techniques for decomposition that *have to be* carried out in modular stated flowsheeting systems *could also be* applied to equation based systems. The effectiveness of tearing in equation based systems has been discussed by Perkins and Sargent (1982).

Netter (1961) has presented an algorithm for finding the non-separable parts of an electrical network. Himmelblau (1967) devised this algorithm for finding disjoint subsets in large sets of algebraic equations, based on boolean operations on the occurrence matrix. Gundersen (1982) implemented this algorithm in PASCAL by using list data structures (occurrence lists) in order to reduce the storage requirements. The 9 disjoint sets in a problem with 90 equations were sorted out in 251·4 msec. When using the 'set'-facility (actually a bit representation), however, the computer time was reduced to 44·0 msec.

The *output set* is a set of pairs defining which variable each equation should be solved with respect to. Finding such a set is a classical problem in several scientific displines. In management science, this set is called the *maximum transversal* (Duff 1981a). Ford and Fulkerson (1962) refer to a maximum transversal as a *maximum assignment*. Researchers in combinatorics use the term *system of distinct representatives*. The bipartite graph is used to describe methods for this task, and the output set is then called a *maximum matching*.

The classical method adopted by chemical engineers is the work of Steward (1962), for which a FORTRAN source is presented by Ledet and Himmelblau (1970). However, substantially faster methods have been published and the latest work is the one of Duff (1981b). Although the algorithm of Hopcroft and Karp (1973) has the lowest worst case complexity of any proposed method to date, it is not necessarily the best method on realistic problems in chemical engineering. Duff has compared it with his own algorithm (MC21A which is part of the Harwell library) on random and structured problems and found the MC21A routine to be superior. A method quite similar to Duff's routine is the ASSIGN ROW algorithm of Gustavson (1976) for which there exists no exact implementation details, no code, and no computer test run times. Duff's algorithm finds the output set for equation sets of up to 100 equations in less than 10 msecs. This compares well with Ledet and Himmelblau's implementation of Steward's method which needs 422·4 and 290·8 msec to find the output set for systems with 88 and 90 equations respectively. (Details on test problems, implementation and computer times are given by Gundersen (1982).)

Given a feasible output set, the adjacency lists required by Tarjan's or Johns' algorithm can be created, and the matrix may be partitioned to obtain a block lower triangular form.

3. Tearing

When the maximal cyclical nets and their elementary cycles have been found, the tearing procedure is a search for a set of cut streams to break all cycles in the flow of mass, energy or information. The tear set is selected on the basis of its cardinality or the weighted sum of elements (weighted tear set).

Rubin (1962) presented the first tearing procedure for chemical plants. By reordering an adjacency type matrix, the algorithm attempts to minimize the total number of unknown parameters in the torn streams.

The later methods for tearing involve one or more of the following operations:

(1) graph simplification,
(2) covering the cycle matrix,
(3) rigorous ordering of the units by enumeration, dynamic programming or branch and bound methods.

Graph simplification is a technique whose purpose is to reduce the graph by the removal of edges and vertices without changing the loop structure and the number of tears in the graph. If an edge is detected that has to be a member of any feasible tear set, this edge is flagged, and the edge is removed from the graph.

In the cover techniques, a set of columns (edges) of the cycle matrix whose union has a non-zero in each row (loop) of the matrix is selected. By removing this set of edges from the graph all cycles are torn and the graph becomes acyclic. The first step of these methods is to reduce the combinatorial problem when having a large number of edges and cycles. Columns and rows are removed by techniques based on the same philosophy as the removal of edges and vertices in graph simplification procedures.

If the simplification rules for the graph or the cycle matrix fail to reduce the graph or the cycle matrix completely, a second step of rigorous ordering is necessary in order to find the minimum tear set.

The cycle matrix was introduced by Steward (1965) as a boolean incidence matrix between loops and streams of a maximal cyclical net. Lee and Rudd (1966) used the cycle rank and stream frequency together with the containment rule in order to simplify the cycle matrix. If the cycle matrix cannot be reduced to an empty matrix, the minimum tear problem becomes the problem of finding the smallest set of columns whose non-zero elements appear in every row of the matrix. This is the classical minimum cover problem. When weighting factors are assigned to each edge, the optimum tearing problem may be stated as an integer linear programming problem (Gupta *et al.* (1974).

Forder and Hutchison (1969) presented an interactive tearing procedure based on the work of Lee and Rudd, where the user is allowed to specify preferred or ineligible tear streams. It is obvious that the interactive nature may substantially reduce the combinatorial problem and also lead to a tear set with better convergence properties than tear sets obtained from a completely automatic routine. The method is used in the process simulation program CONCEPT from Computer Aided Design Centre (CADC).

Sargent and Westerberg (1964) revealed two very simple properties of graphs, the simply linked unit and the two-way link, which can be used to simplify the graph. They introduced dynamic programming to solve the second, rigorous step when graph simplification fails to reduce the graph completely.

Christensen and Rudd (1969) generalized Sargent and Westerberg's simply linked unit in the concept of ineligible edges. The algorithm of Christensen and Rudd does not guarantee a minimum number of tears, but a close to optimal ordering is attained by heuristic rules when graph simplification cannot reduce the graph completely.

Barkley and Motard (1972) suggested a different variation to the graph simplification technique by performing an interval reduction to a signal flow graph. The second stage of rigorous ordering of the non-empty graph after simplification is not discussed, since all their test problems was reduced completely. Besides, this method is limited to problems with only unit weightings (cardinality only).

While Sargent and Westerberg use permutations of vertices (unit orderings) in their state space representations for dynamic programming, Upadhye and Grens (1972) use combinations of cycles opened. Dynamic programming reduces the combinatorial problem when compared with exhaustive enumeration, but this is achieved at the expense of severe storage requirements. The number of states required by these two methods are $2^{n'}$ for Sargent and Westerberg's algorithm and 2^c for Upadhye and Grens's algorithm where n' is the number of vertices in the reduced graph after the first simplification steps and c is the number of elementary cycles. This limits the application of dynamic programming to 15–20 cycles or vertices of the reduced graph. It will be shown later that most of the classical graphs from decomposition articles in the chemical engineering literature involve *more* than 20 cycles.

The number of vertices in the reduced graph depends on the simplification rules available and the structure of the graph and is therefore difficult to predict. The two simple rules presented by Sargent and Westerberg managed to reduce a graph of 19 vertices, 31 edges and 20 cycles all in one maximal net to a graph with only four supernodes.

In this case the algorithm of Upadhye and Grens would have to deal with an enormous state-space ($2^{20}=1048576$) and the algorithm of Sargent and Westerberg ($2^4=16$) appears to be superior. The Upadhye and Grens method thus seems to be interesting only for graphs with a large number of vertices and a moderate number of cycles. In this case, however, the graph simplification of Sargent and Westerberg may go a long way towards a complete elimination of the graph.

On the other hand, Cheung and Kuh (1974) have pointed out that the size of the cycle matrix may be substantially reduced if only the pertinent elementary cycles are generated. These are the cycles that are pertinent for determining the cardinality of the minimum tear set (index of the graph).

A more thorough study is needed to find out whether graph simplification and the concept of pertinent cycles may reduce the storage requirements of dynamic programming enough to deal with all interesting problems of arbitrary complexity. The limit of 15–20 supernodes or pertinent cycles appears, however, to be a severe limitation of these methods.

In order to overcome the inherent dimensionality difficulty associated with dynamic programming, Pho and Lapidus (1973) applied a branch and bound method to obtain the minimum tear set after a first step of graph simplification. The branch and bound technique is a clever enumeration scheme with the advantage that through the application of bounds, only a small fraction of the possible solutions needs actually be examined. A necessary condition for a successful application of the branch and bound method is that the lower and upper bounds are evaluated efficiently and that these values are reasonable.

Another important part of branch and bound methods is the strategy for branching. This is pointed out by Johns and Müller (1976) who have presented an algorithm whose major difference from the algorithm described by Pho and Lapidus is in the branching strategy. The Pho and Lapidus algorithm branches by making a stream either recycle or feed-forward. Although removing a potential recycle stream makes good progress towards the solution because only 5–10 % of the streams are recycles in an optimal sequence, tagging a feed-forward stream makes only marginal progress since the majority of streams *are* feed-forward. Besides, since the branching stream is selected fairly arbitrarily, there is a high probability that it is *not* a recycle stream.

Johns and Müller attempt instead to tag recycle streams on both branches, such that one of the two streams must be included in the optimal tear set. Since they succeed in this for about 80 % of the branching points, this assures a more rapid progress towards the optimal solution.

Motard and Westerberg (1981) have presented an implicit enumeration (branch and bound) algorithm for the minimum weighted tear problem which is based on the work of Upadhye and Grens (1975). Applying direct substitution to a simple graph for different tear sets, Upadhye and Grens illustrated that the difference in convergence behaviour is due to the *unit ordering* corresponding to the tear sets. Different tear sets that lead to the same unit ordering have the same convergence properties. Selecting tear sets from a 'non-redundant' family gives decomposed problems with better convergence properties. Being the flowsheet tearing algorithm for the ASPEN program (MIT), Motard and Westerberg's method has been tested on hundreds of practical problems. However, since the logical steps of the algorithm use the loop/incidence matrix (cycle matrix), it appears not to be able to deal with complex problems like the heavy water plant presented in this paper.

The overall reason for using decomposition algorithms is to reduce the computer *time* and *storage* requirements. In order to achieve this it is important to use both the structural and the numerical information in the search for an 'optimal' tear set. Genna and Motard (1975) thus selects the optimum tear set *adaptively* as the recycle computation proceeds. The convergence acceleration in achieving mass and energy balance depends on the properties of the jacobian matrix of the linearized process. The matrix with the smallest maximum eigenvalue identifies the optimum tear set. Since eigenvalue calculation is a tedious task, Genna and Motard use the sum of squares of the diagonal elements as an approximate criterion.

The numerical properties of the decomposed system are also discussed by Westerberg and Edie (1971) and Kevorkian and Snoek (1973). In the search for an optimum tear set, the numerical information may be put into the weighting factors for each stream in the process.

Also in electrical engineering the literature abounds with algorithms to find the minimum essential set (tear set) of directed graphs. Some of the significant ones are the methods of Divieti and Grasseli (1968), Guardabassi (1971), Cheung and Kuh (1974), Smith and Walford (1975), Guardabassi and Sangiovanni-Vincentelli (1976) and Kevorkian (1980). Unfortunately, these algorithms are purely minimum cardinality methods, without any weighting factors associated with each edge. Nevertheless, several of the techniques are similar to those found in the chemical engineering literature and the algorithms are here also divided in two basic classes: Topological methods (graph simplification) and cover methods (cycle matrix).

Guardabassi (1971) has presented a completely topological algorithm based on the McClusky (1965) method for solving the prime implicant problem of switching theory. Guardabassi's method involves a first stage of graph simplification and a rigorous stage using a branch and bound technique to reduce the graph completely. The *weighted* tear problem, however, is not solved.

Cheung and Kuh (1974) have presented a method which involves (1) graph simplification, (2) generation of the pertinent cycles and (3) covering the resulting cycle matrix.

In a very comprehensive work in this field, Kevorkian (1980) shows that it is possible to find an essential set (tear set) whose cardinality is within a known tolerance of the optimum value in $O(n^2)$ computation steps. A detailed formulation of the algorithm is given by Kevorkian in Pidgin Algol.

A new non-optimal procedure

Gundersen (1982) has presented another non-optimal routine for the weighted case of tearing which has a very simple formulation. In order to solve a problem which has been stated to be NP-complete (tearing), this routine takes advantage of the fact that partitioning procedures are extremely fast and have a linear complexity function.

The method which will be briefly discussed here breaks down the directed graph sequentially from its maximal cyclical nets to a feed-forward (acyclic) graph. Between each application of the partitioning routine, all entering edges to a selected vertex are removed (torn). The rule for selecting tears is illustrated below where the vertex j with all entering and leaving edges are drawn.

The sum of weightings for the entering edges of vertex j is found by:

$$\rho(j) = \sum_{i=1}^{r} w_{ij}^{+}$$

The sum of weightings for the leaving edges of vertex j is similarly found by:

$$\tau(j) = \sum_{i=1}^{s} w_{ij}^{-}$$

The key variable for each vertex in the selection procedure is defined as the ratio between the weighted sum of entering and the weighted sum of leaving edges:

$$\alpha(j) = \rho(j)/\tau(j)$$

In the proposed method, all entering edges (inputs) to the vertex with the *smallest* $\alpha(j)$ are selected as tears. The philosophy of this heuristic rule is that by guessing few input variables to a unit, many ouput variables become known and the yield of information is maximized. Since only local information of the graph is used to select the tears and there are no optimization steps, the method clearly cannot guarantee optimality and its usefulness has to be evaluated by running typical problems.

Since, however, optimal routines for tearing exist, the non-optimal routines have no interest as stand-alone programs unless the problem is too large to be handled with optimal routines in a reasonable amount of computer time. If the heuristic methods are sufficiently efficient, however, they may be used to find bounds in branch and bound strategies.

The steps of the proposed method are:

(1) Read the adjacency structure of the graph from a file and create lists of predecessors (PREDLIST) and successors (SUCCLIST) for each vertex.

(2) Initialize variables used in Tarjan's routine for finding the strongly connected components (SCC) of a graph (partitioning). A pointer array (POINTER) is used to store the addresses of the start and end of each SCC in the sequence list (SEQ) produced by Tarjan's routine.

(3) Apply Tarjan's algorithm STRONGCONNECT (given in detail by Tarjan (1972) and thus not discussed here) on the original graph. The number of SCCs detected by this routine is stored in the variable NSCC.

(4) Carry out the following substeps as long as the number of strong components is less than the number of vertices (NSCC < NV). When all loops in the graph are torn, NSCC becomes equal to NV (each strong component involves a single vertex only).

(i) Go through the pointer array POINTER and store single vertices (vertices that are not involved in loops) in the final sequence array FINSEQ. Flag these placements in the boolean array STORED.

(ii) Find the first strong component of array SEQ with *more* than one vertex. Compute the above defined variable α of each vertex of this SCC. It is important in this calculation *not* to take into account vertices from *other* SCCs. This is achieved by assigning the variable NSCC which is monotonously increasing to the array CURR for the vertices of the current SCC.

When computing $\rho(j)$ and $\tau(j)$ for each vertex only the SUCCLIST of the vertices in the current SCC are inspected. Vertices on these lists with CURR-values *less* than NSCC are neglected.

The vertex with the smallest value of α (say vertex k) is flagged.

(iii) Remove all entering edges (inputs) to vertex k. This is achieved by going through the SUCCLISTs of all vertices and deleting (bypassing) vertex k on these lists.

(iv) Initialize variables for Tarjan's partitioning procedure.

(v) Apply Tarjan's algorithm STRONGCONNECT to find the new (and smaller) SCCs of the graph.

(vi) Return to (i) if NSCC is less than NV.

(5) Store the single vertices from the last application of Tarjan's algorithm in the final sequence array FINSEQ.

(6) Go through the PREDLISTs of the vertices in the final sequence (FINSEQ). For each vertex encountered in FINSEQ, the corresponding element of the boolean array COMP is assigned the value 'true'. If, when going through the list of predecessors (PREDLIST) for vertex i, a vertex j is encountered which has still not been 'computed' (COMP(j) = 'false'), the edge from j to i has to be torn and is thus printed in the output file.

A computer listing (PASCAL) of the proposed method (TEARGRAPH) is given in Appendix B.

4. Problems

The group of test problems involves the classical graphs from the chemical engineering literature and in addition the very complex graph of a heavy water plant on which tearing by hand is impossible. With 109 vertices, 163 edges and 13746 elementary cycles this problem should be posed as a bench-mark for new tearing procedures. It is important to note that this problem is not a constructed, hypothetical one since a computerized mass and energy model for this process exists in Norsk Hydro. Thus there is a need for automatic routines that are able to find the minimum (cardinality of weighted sum) tear set for graphs of this complexity. The graphs are presented in Appendix A and Table 1 gives the characteristics of each problem such as the number of vertices, edges, maximal cyclical nets and elementary cycles.

Table 1. Complexity of test problems.

Problem	Dimension (number of)				
No.	Vertices	Edges	Max. nets	Cycles	References
1	6	36	1	415	Complete graph
2	12	21	1	22	Pho/Lapidus
3	15	35	3	27	Barkley/Motard
4	19	31	1	20	Sargent/Westerberg
5	25	32	1	10	Christensen/Rudd ('first')
6	29	37	1	11	Jain/Eakman (HF-alkylation)
7	30	42	1	31	Christensen/Rudd ('second')
8	41	61	1	103	Shannon (Sulfuric acid)
9	50	79	1	22	Jain/Eakman (Vegetable oil)
10	109	163	1	13746	Gundersen (Heavy water)

The graph of the heavy water plant (Fig. 10 of Appendix A) has a very complex loop structure and involves 13746 elementary cycles all in one maximal net. Tearing procedures based on manipulation with the cycle matrix cannot handle this problem and it is hoped that this problem is adopted by authors of papers on decomposition in the future as a bench-mark for new and existing algorithms.

5. Results

PASCAL has proved to be an efficient tool and all implementations has been done in this language. With the special features in PASCAL for list processing ('records' and 'pointers'), the graph theoretic based algorithms are easily programmed and the resulting codes are much easier to read and interpret than similar FORTRAN versions. The concept of linked lists is applicable to many problems, and it is especially suitable for sparse systems. Thus the linked lists are used not only in the analysis, but also in the elimination part of matrix inversion (equation solving).

Some of the methods for partitioning have been implemented and run on the 10 test problems. The results are shown in Table 2. The matrix multiplication method to find the reachability matrix is operating according to

$$R=(A+I)^{n-1}$$

with successively squaring the matrix until the power is equal to or greater than $n-1$. A simultaneous row matrix multiplication scheme (Jennings (1977) has been used to take advantage of the sparsity.

Table 2. Partitioning. Computer times (msec).

Problem No.	Matrix mult.	Warshall	Thorelli	Tarjan SCC	Tarjan EC	Duff and Reid	Johns
1	22·8	6·0	15·8	3·6	79·2	1·0	1·0
2	143·2	36·0	99·8	3·8	17·2	1·6	1·4
3	228·4	63·0	66·8	6·4	29·8	2·2	1·8
4	598·4	89·6	596·0	6·4	22·4	2·4	2·2
5	1063·0	207·8	1729·8	7·6	22·2	3·0	2·4
6	1346·6	278·2	1300·8	11·2	18·8	3·4	2·8
7	1806·6	360·8	3560·6	9·6	71·8	3·4	3·0
8	7564·2	1131·6	16379·4	15·8	221·0	4·8	4·2
9	6074·8	905·0	875·8	26·8	93·6	6·2	4·4
10	100547·6	6547·8	> 5 min	55·8	34127·6	12·8	10·8

The implementation of Warshall's algorithm operates on computer *words* and the possibility of using *bit* representation is thus not taken into account. In the implementation of Thorelli's algorithm the result strongly depends on how the list of pairs (paths) is handled. One cannot, however, avoid that the time used to check for previous appearances of pairs grows rapidly, and the method is not able to compete with Warshall's very simple although $O(n^3)$ algorithm.

Tarjan SCC and EC are the implementations made by Gundersen in PASCAL of Tarjan's methods for finding the strongly connected components (partitions) and the elementary cycles. These programs both use recursion and this is the major reason why the SCC program executes slower than the FORTRAN version of Tarjan's method presented by Duff and Reid (1976). Table 2 also shows that Johns' method implemented in FORTRAN executes slightly faster than Duff and Reid's FORTRAN version of the later and far better known Tarjan algorithm.

Tarjan's algorithm for detecting the elementary cycles of a digraph was also tested on a complete graph with 9 vertices and 125673 cycles. Tsukiyama *et al.* solved this problem in 10·51 sec on an IBM 370/165, while Tarjan reported 101·20 sec on an IBM 360/65. With our recursive implementation this problem was solved in 34·34 sec on a UNIVAC-1162 (Tiernan's algorithm took 219 sec on a Burroughs 5500 computer to solve a complete graph with only 8 vertices). Since Tsukiyama *et al.* obviously has not tested their algorithm against a non-recursive implementation of Tarjan's method, it is still an open question which method is the fastest one.

In the presentation of methods for tearing, some evaluations were made based on a discussion of the *idea* adopted by the different algorithms. In this work only a few methods have been available and actually run on the test problems. In addition to the new non-optimal method presented, the NETWORK program of Jain and Eakman has been used.

The algorithm of Jain and Eakman breaks down and is unable to solve the heavy water plant because it has to store and operate on 13746 elementary cycles. Also other methods that use the cycle matrix cannot handle problems of this complexity.

Table 3 shows that the presented, non-optimal routine for the weighted case of tearing is very fast and finds a number of tears that is surprisingly close to the optimum number. The heavy water plant is solved in 721·6 msec and the number of selected tears (13) is only one more than the optimum number (12). The routine uses a recursive version of Tarjan's method repeatedly for partitioning. Since the partitioning part of the routine is significant (about 70%) and Duff and Reid's non-recursive implementation of Tarjan's method is about 3 times faster than Gundersen's recursive version, the computer times in Table 3 may be substantially reduced by avoiding recursion. The fact that the proposed method is very close to optimal and executes fast makes it a possible upper bound algorithm in a branch and bound strategy.

Table 3. Tearing. Computer times and number of tears.

Problem No.	Dimension		Min. No. of tears	Proposed method		NETWORK	
	Vertices	Cycles		msec	Tears	msec	Tears
1	6	415	21	30·2	21	400·0	‡
2	12	22	2	21·2	3	265·2	2
3	15	27	6	50·6	7		†
4	19	20	6	44·2	6	268·6	6
5	25	10	3	35·8	3	227·2	3
6	29	11	5	53·2	5	189·2	5
7	30	31	3	59·6	4	639·8	3
8	41	103	5	97·6	5	1628·0	‡
9	50	22	8	130·0	8	302·2	8
10	109	13746	12	721·6	13		§

† NETWORK (local version) is not able to handle more than one maximal cyclical net.

‡ NETWORK (local version) has an upper limit on the number of cycles (75).

§ NETWORK is based on an idea unable to solve problems of this complexity (too many cycles).

Johns and Müller's branch and bound method as it is implemented in Flowpack II finds the *optimum* tear set for the heavy water plant in 33 sec on a DEC-10. According to Johns (1981) there are several improvements that could speed up the algorithm. These improvements include a better graph simplification routine and a better routine for locating small cycles ('triangles' etc.). Very recently, Johns (1982) has presented an edge re-weighting method which provides a close lower bound on the optimal tearing which operates in computer times related to the square of the problem size. This algorithm is together with Kevorkian's method the only known method for obtaining *lower* bounds on the number of tears in a digraph. Kevorkian's method, however, cannot handle weighting factors in the tearing.

6. Conclusion

The path tracing methods have been found to be superior to the boolean matrix methods for partitioning. Johns' algorithm has been evaluated to run slightly faster than the later and far better known algorithm of Tarjan. These methods are almost identical in idea, but use different techniques in the book-keeping. Test runs support that they both execute in computer times that are linear functions of the number of vertices in the graph.

Methods for tearing based on branch and bound techniques are found to be superior to algorithms based on enumeration and dynamic programming. The Pho and Lapidus algorithm has not been evaluated, but Johns and Müller claim to branch in a way that reduces the number of states to be examined. Their method as it is implemented in Flowpack II finds the optimum tear set for the large and very complex graph of a heavy water plant with 13746 elementary cycles in half a minute on a DEC-10 computer.

Use of the cycle matrix puts a severe limit on tearing procedures. It is not the generation of the elementary cycles that is prohibitive (Tarjan's method enumerates the 13746 cycles of the heavy water plant in 34 sec on a UNIVAC-1162), but rather the size of this matrix (=2·24 million storage locations for the heavy water plant).

The proposed non-optimal algorithm for tearing may serve as an efficient upper bound routine in a branch and bound method. The algorithm is based on the idea of a repeated application of a partitioning procedure (Tarjan or Johns). Between each application, all inputs to a selected vertex are removed (torn). Although the tears are selected according to heuristic (local) rules, the algorithm finds the optimum number of tears for 60% of the test problems, and the number of tears found for the remaining problems is only one more than the optimum number.

As an algorithm for calculating upper bounds in a branch and bound strategy, the proposed method has the following important features:

—It finds a number of tears that is surprisingly (regarding its simplicity) close to the optimum number.

—It solves the minimum weighted tear problem equally easily as the minimum cardinality problem.

—It is very fast, since a graph with 109 vertices, 163 edges and 13746 elementary cycles is torn in less than a second on a UNIVAC 1162.

—It operates slightly faster than $O(V, E)^2$.

References

Baker, J. J. (1962). A note on multiplying boolean matrices. *Comm. of the ACM*, **5**, 102.

Barkley, R. W., and Motard, R. L. (1972). Decomposition of nets. *Chem. Eng. J.*, **3**, 265–275.

Berge, C. (1962). *The theory of graphs and its applications* (John Wiley & Sons, New York).

Berger, F., and Perris, F. A. (1979). FLOWPACK II—a new generation of systems for steady state process flowsheeting. *Comp. & Chem. Eng.*, **3**, 309–317.

Billingsley, D. S. (1967). Letter to editor. *Chem. Eng. Sci.*, **22**, 719.

Bowie, W. S. (1976). Applications of graph theory in computer systems. *Int. J. of Comp. and Inf. Sci.*, **5**, 9–31.

Cheung, L. K., and Kuh, E. S. (1974). The bordered triangular matrix and minimum essential sets of a digraph. *IEEE Trans. Circ. Sys.*, **CAS-21**, 633–639.

Christensen, J. H., and Rudd, D. F. (1969). Structuring design computations. *AIChE J.*, **15**, 94–100.

Dahl, O.-J., Myhrhaug, B., and Nygaard, K. (1968). *SIMULA 67—common base language*, publ. no. S-2 (Norwegian Computing Center, Oslo).

Divieti, L., and Grasseli, A. (1968). On the determination of minimum feedback arc and vertex sets. *IEEE Trans. Circ. Theory*, **CAT-15**, 86–89.

Duff, I. S. (1981a). On algorithms for obtaining a maximum transversal. *ACM Trans. on Math. Software*, **7**, 315–330.

Duff, I. S. (1981b). Algorithm 575. Permutations for a zero-free diagonal. *ACM Trans. on Math. Software*, **7**, 387–390.

Duff, I. S., and Reid, J. K. (1976). *An implementation of Tarjan's algorithm for the block triangularization of a matrix*. AERE Harwell report CSS29.

Ford, L. R., and Fulkerson, D. R. (1962). *Flows in networks* (Princeton University Press).

Forder, G. J., and Hutchison, H. P. (1969). The analysis of chemical plant flowsheets. *Chem. Eng. Sci.*, **24**, 771–785.

Genna, P. L., and Motard, R. L. (1975). Optimal decomposition of process networks. *AIChE J.*, **21**, 4, 656–663.

Guardabassi, G. (1971). A note on minimal essential sets. *IEEE Trans. Circ. Theory*, **CAT-18**, 557–560.

Guardabassi, G., and Sangiovanni-Vincentelli, A. (1976). A two level algorithm for tearing. *IEEE Trans. Circ. Sys.*, **CAS-23**, 783–791.

Gundersen, T. (1982). Decomposition of large scale chemical engineering systems. Dr. Ing. thesis, Dep. of Chem. Eng., Univ. of Trondheim, Norway.

Gupta, P. K., Westerberg, A. W., Hendry, J. E., and Hughes, R. R. (1974). Assigning output variables to equations using linear programming. *AIChE J.*, **20**, 397–399.

Gustavson, F. G. (1976). Finding the block lower trangular form of a sparse matrix, in Bunch J. R. and Rose D. J. (eds) *Sparse matrix computations*, pp. 275–289 (Academic Press, New York).

Harary, F. (1962). A graph theoretic approach to matrix inversion by partitioning. *Num. Math.*, **4**, 128–135.

Harary, F. (1969). *Graph theory* (Addison-Wesley, Reading, Mass.).

Himmelblau, D. M. (1966). Decomposition of large scale systems—1. Systems composed of lumped parameter elements. *Chem. Eng. Sci.*, **21**, 425–438.

Himmelblau, D. M. (1967). Decomposition of large scale systems—2. Systems containing nonlinear elements. *Chem. Eng, Sci.*, **22**, 883–895.

Hlaváček, V. (1977). Analysis of a complex plant—steady state and transient behaviour (Journal review). *Comp. & Chem. Eng.*, **1**, 75–100.

Hopcroft, J. E., and Karp, R. M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, **2**, 225–231.

Jain, Y. V. S., and Eakman, J. M. (1971). Identification of process flow networks. *AIChE 68th meeting, Houston, Texas*, Feb. 1971.

Jennings, A. (1977). *Matrix computations for engineers and scientists* (John Wiley & Sons).

Jensen, K., and Wirth, N. (1978). *PASCAL—user manual and report*, 2nd ed. (Springer Verlag).

Johns, W. R. (1970). Mathematical considerations in preparing general purpose computer programs for the design or simulation of chemical processes. *European Confed. of Chem. Eng. Conf.*, Florence, Italy, April 1970.

Johns, W. R. (1981). Dep. of Chem. Eng., Polytechnic of the South Bank, London, Personal communication, Nov.

Johns, W. R. (1982). Re-weighting algorithms for tearing graphs representing process flowsheets. *Inst. Chem. Engrs. Jubilee Symp.*, London, April 5–7.

Johns, W. R., and Müller, F. Optimal sequence for computer flowsheeting calculations, Technisch-Chemisches Labor, Eidgenossisches Technische Hochschule, Zürich, 1976.

KARP, R. M. (1972). Reducibility among combinatorial problems in Miller R. E. and Thatcher J. W. *Complexity of computer computations*, pp. 85–104 (Plenum Press, New York).

KEHAT, E., and SHACHAM, M. (1973). Partitioning and tearing of system flowsheets. *Proc. Techn.*, **18,** 115–118.

KEVORKIAN, A. K. (1980). General topological results on the construction of a minimum essential set of a directed graph. *IEEE Trans. Circ. Sys.*, **CAS-27,** 293–304.

KEVORKIAN, A. K., and SNOEK, J. (1973). Decomposition in large scale systems. Theory and applications of structural analysis in partitioning, disjointing and constructing hierarchical systems, in Himmelblau D. M. (ed) *Decomposition of large scale problems* (Elsevier).

LEDET, W. P., and HIMMELBLAU, D. M. (1970). Decomposition procedures for the solving of large scale systems, from *Advances in Chemical Engineering*, pp. 185–254.

LEE, W., and RUDD, D. F. (1966). On the ordering of recycle calculations. *AIChE J.*, **12,** 1184–1190.

MAH, R. S. H. (1974). A constructive algorithm for computing the reachability matrix. *AIChE J.*, **20,** 1227–1228.

MCCLUSCKY, E. J. (1965). *An introduction to the theory of switching circuits* (McGraw-Hill, New York).

MOTARD, R. L., and WESTERBERG, A. W. (1981). Exclusive tear sets for flowsheets. *AIChE J.*, **27,** 725–732.

MUNRO, I. (1971). Efficient determination of the transitive closure of a directed graph. *Information Proc. Letters*, **1,** 56–58.

NETTER, Z. (1961). Graphs and directed sum matrices. *I.R.E. Trans. on Circ theory*, **CT-8,** 77–78.

NORMAN, R. L. (1965). A matrix method for location of cycles of a directed graph. *AIChE J.*, **11,** 450–452.

PERKINS, J. D., and SARGENT, R. W. H. (1982). SPEEDUP: A computer program for steady-state and dynamic simulation and design of chemical processes. *AIChE Symp. Ser.*, **78,** 1–11.

PHO, T. K., and LAPIDUS, L. (1973). An optimum tearing algorithm for recycle systems. *AIChE J.*, **19,** 1170–1181.

PURDOM, P. (1970). A transitive closure algorithm. *BIT*, **10,** 76–94.

RUBIN, D. I. (1962). Generalized material balance. *C.E.P. Symp. Ser.*, **58,** 54–61.

SARGENT, R. W. H., and WESTERBERG, A. W. (1964). SPEEDUP in chemical engineering design. *Trans. Inst. Chem. Eng.*, **42,** T190–T197.

SMITH, G. W., and WALFORD, R. B. (1975). The identification of a minimal feedback vertex set of a directed graph. *IEEE Trans. Circ. Sys.*, **CAS-22,** 9–15.

STEWARD, D. V. (1962). On an approach to techniques for the analysis of the structure of large systems of equations. *SIAM Review*, **4,** 321–342.

STEWARD, D. V. (1965). Partitioning and tearing systems of equations. *J. SIAM*, *Numer. anal.* (ser. B,) **2,** 345–365.

TARJAN, R. (1972). Depth first search and linear graph algorithms. *SIAM J. Computing*, **1,** 146–160.

TARJAN, R. (1973). Enumeration of the elementary circuits of a directed graph. *SIAM J. Computing*, **2,** 211–216.

THORELLI, L. E. (1966). An algorithm for computing all paths in a graph. *BIT* **6,** 347–349.

TIERNAN, J. C. (1970). An efficient algorithm to find the elementary circuits of a graph. *Comm. of the ACM*, **13,** 722–726.

TSUKIYAMA, S., SHIRAKAWA, I., and OZAKI, H. (1975). An algorithm for generating the cycles of a digraph. *Electronics and Communications in Japan*, **58-A,** 8–15.

UPADHYE, R. S., and GRENS, E. A. II (1972). An efficient algorithm for optimum decomposition of recycle systems. *AIChE J.*, **18,** 533–539.

UPADHYE, R. S., and GRENS, E.A., II (1975). Selection of decomposition for chemical process simulation. *AIChE J.*, **21,** 136–143.

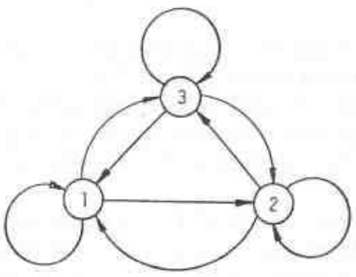
WARSHALL, S. (1962). A theorem on boolean matrices. *J. of the ACM*, **9,** 11–12.

WEINBLATT, H. (1972). A new search algorithm for finding the simple cycles of a finite directed graph. *J. of the ACM*, **19,** 43–56.

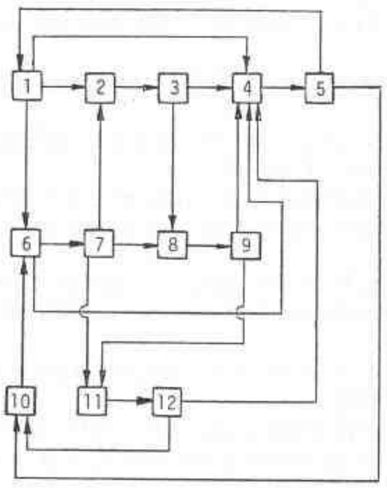
WESTERBERG, A. W., and EDIE, F. C. (1971). An approach to convergence and tearing in the solution of sparse equation sets. *Ch. Eng. J.*, **2,** 17–24.

Appendix A

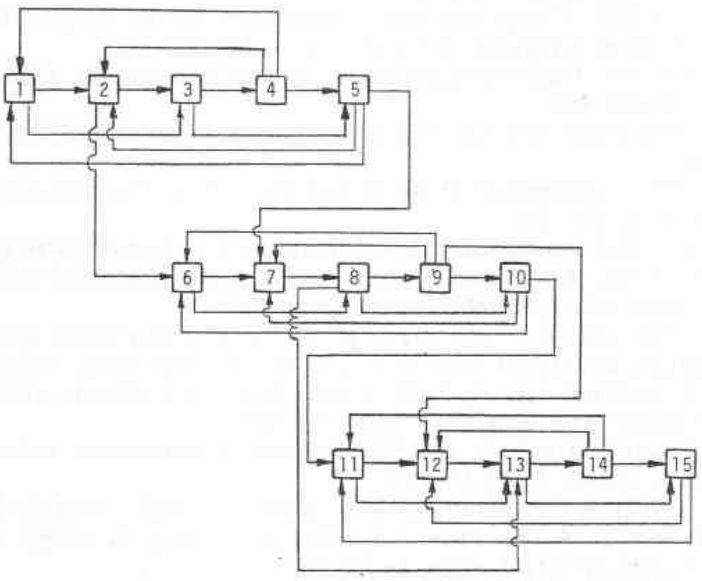
Directed graphs for the test problems.



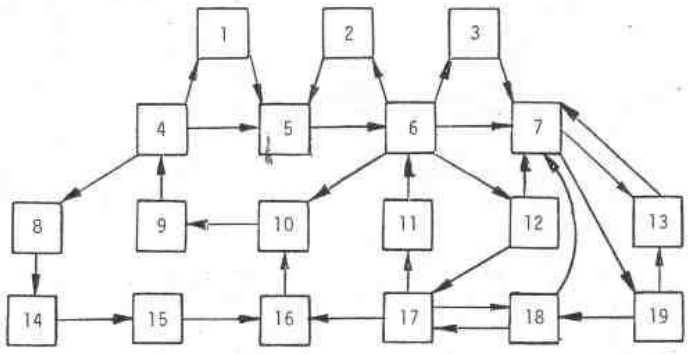
Problem No. 1.



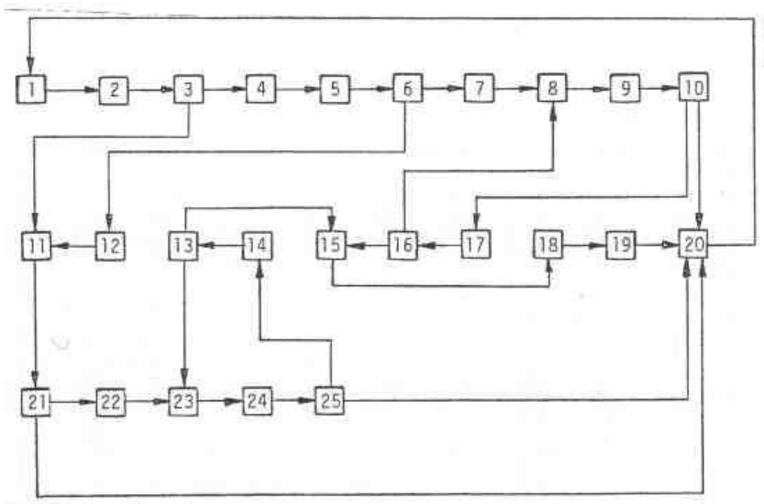
Problem No. 2.



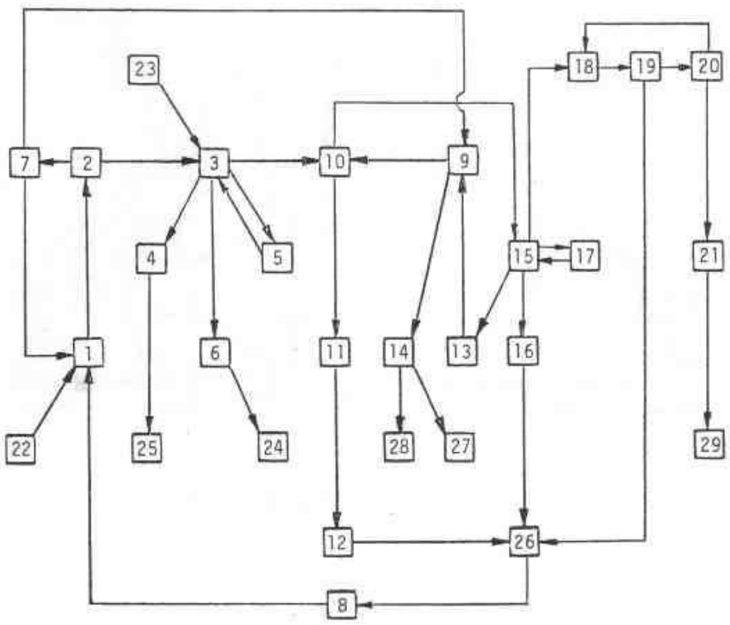
Problem No. 3.



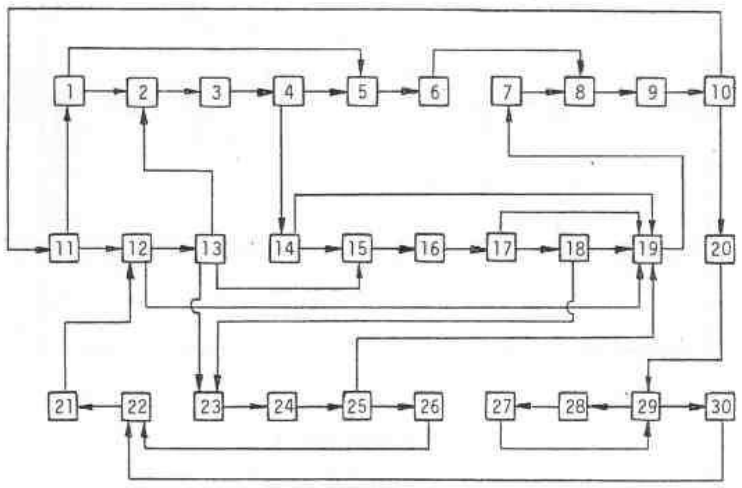
Problem No. 4.



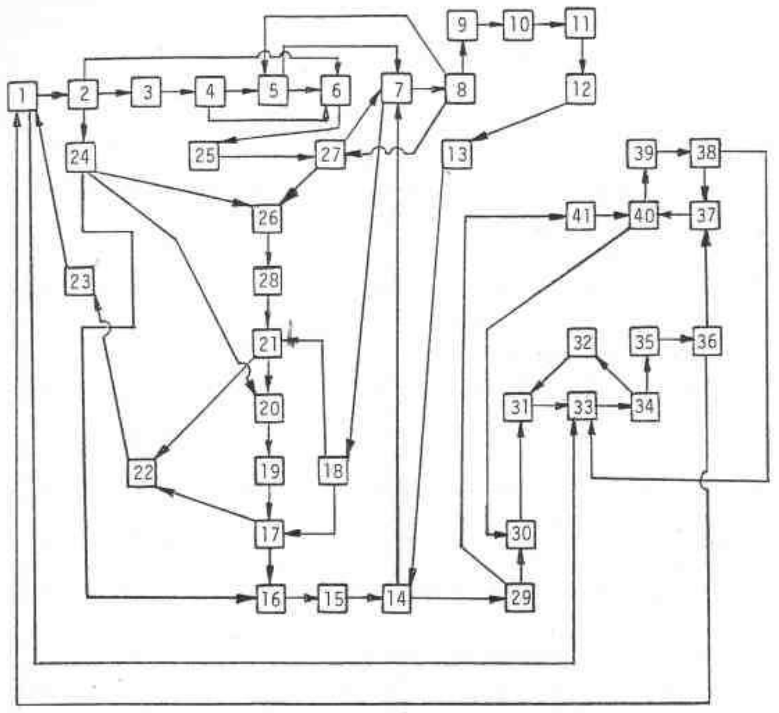
Problem No. 5.



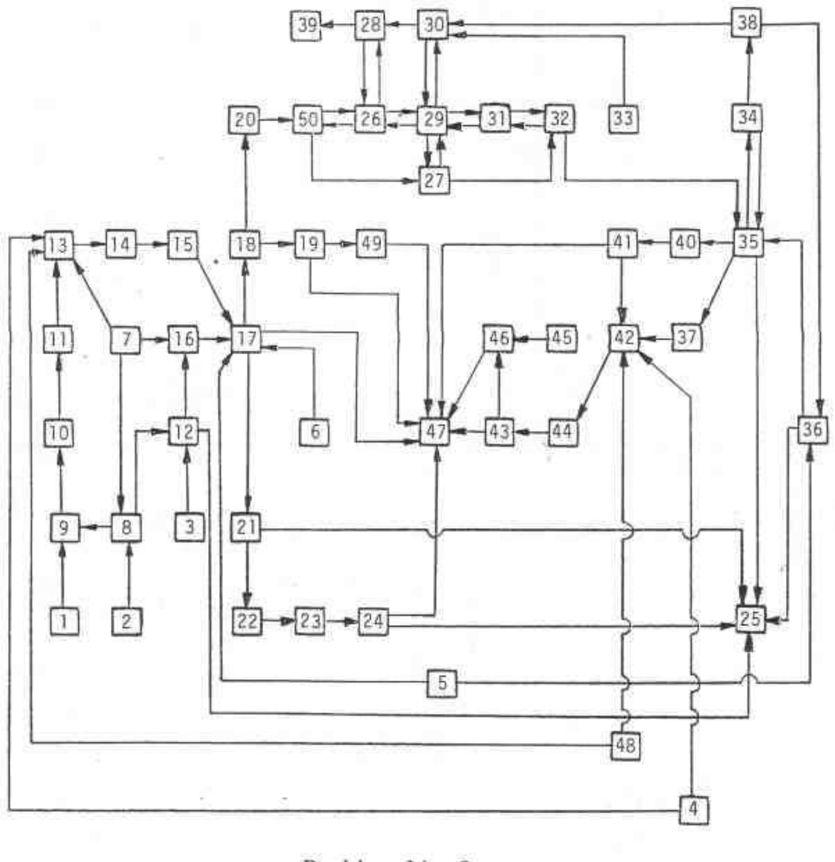
Problem No. 6.



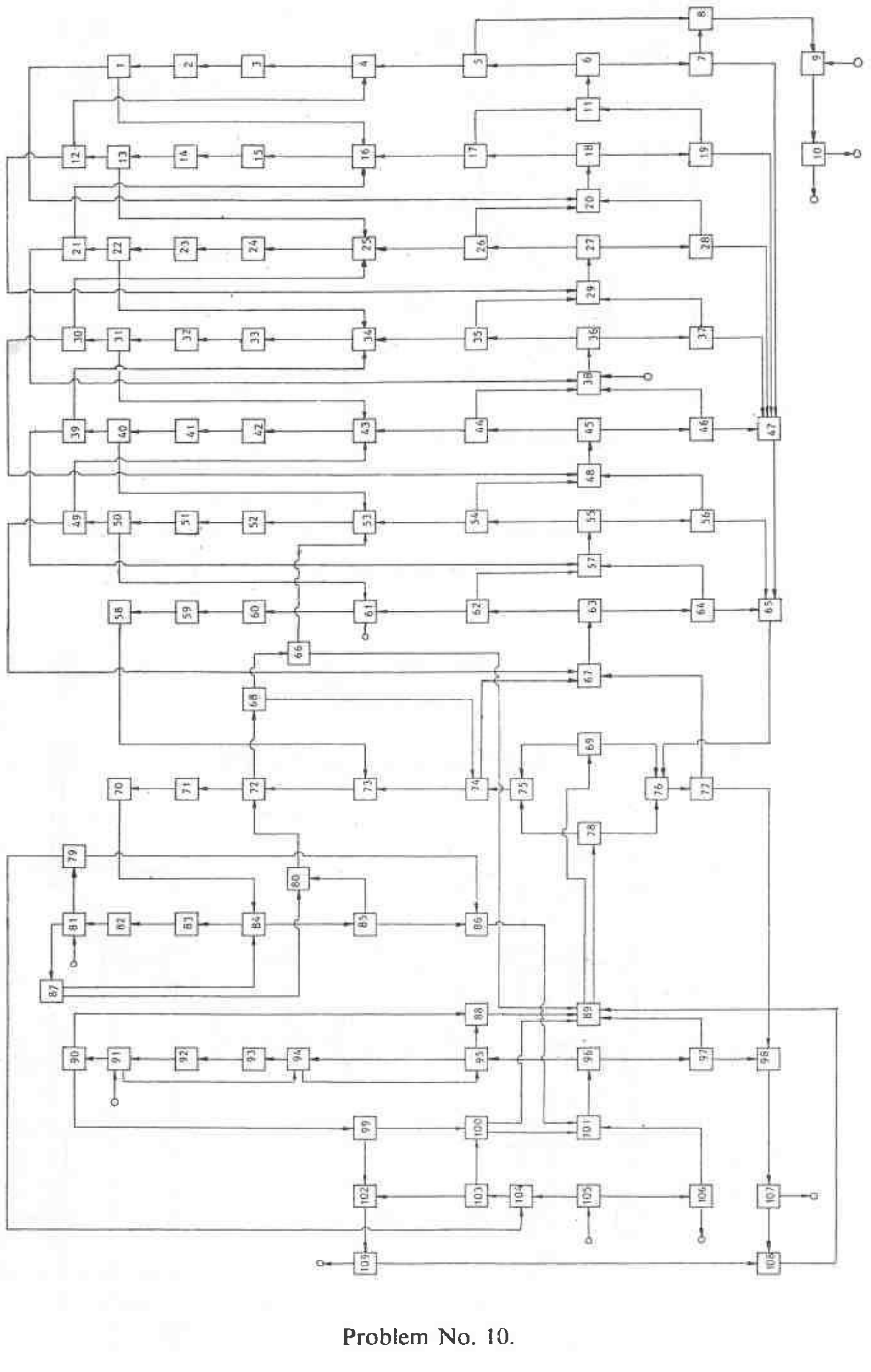
Problem No. 7.



Problem No. 8.



Problem No. 9.



Problem No. 10.

Appendix B

Computer listing of the proposed method in PASCAL, as implemented on a UNIVAC computer.

```
(*$R20*)
PROGRAM TEARGRAPH(INPUT,OUTPUT);

(*  ALGORITHM FOR TEARING BASED ON A REPEATED  *)
(*  APPLICATION OF TARJAN'S ALGORITHM FOR      *)
(*  FINDING AND PRECEDENCE-ORDERING THE STRONG *)
(*  COMPONENTS OF A DIRECTED GRAPH.            *)

(*  BETWEEN EACH APPLICATION ALL ENTERING      *)
(*  EDGES OF A SELECTED VERTEX ARE TORN.       *)
(*  THIS PROCEDURE CONTINUES UNTIL EACH        *)
(*  STRONG COMPONENT CONSISTS OF ONLY ONE      *)
(*  VERTEX, I.E. ALL LOOPS ARE OPENED AND      *)
(*  THE TORN GRAPH IS ACYCLIC.                 *)

(*  AUTHOR: T. GUNDERSEN, NTH, CHEM. ENG. 81   *)

LABEL 300;
CONST N=109;
TYPE  LINK =^ NODE;
      NODE = RECORD
               NR : INTEGER;
               WT : REAL;
               NEXT : LINK;
             END;
      VEC = ARRAY(.1..N.) OF INTEGER;
VAR   SUCCLIST,PREDLIST,PREDPOINT : ARRAY (.1..N.) OF LINK;
      LOWLINK,NUMBER,STACK,SEQ,FINSEQ,POINTER,CURR : VEC;
      RO,TAU,ALFA : ARRAY(.1..N.) OF REAL;
      COMP,STORED : ARRAY(.1..N.) OF BOOLEAN;
      PP,CURRENT,PREVIOUS,THISNODE : LINK;
      NV,I,J,K,KK,NSCC,COUNT,NSTORE,L,NSTACK,START,IEND : INTEGER;
      T1,T2,T3,W,MIN : REAL;

PROCEDURE CLTIME(VAR T:REAL);FORTRAN;

PROCEDURE STRONGCONNECT(V : INTEGER);

   (*  RECURSIVE VERSION OF TARJAN'S ALGORITHM  *)
   (*  THE PRECEDENCE ORDERED SEQUENCE WILL BE  *)
   (*  STORED IN A PROPER ORDER IN VECTOR SEQ.  *)
```

```
 LABEL 100,200;
 VAR   W : INTEGER;
       THATNODE : LINK;
 BEGIN
    I:=I+1;
    LOWLINK(.V.):=I;
    NUMBER(.V.):=I;
    NSTACK:=NSTACK+1;
    STACK(.NSTACK.):=V;
    THATNODE:=SUCCLIST(.V.);
    WHILE THATNODE^.NEXT <> NIL DO
    BEGIN
       THATNODE:=THATNODE^.NEXT;
       W:=THATNODE^.NR;
       IF NUMBER(.W.)=0 THEN
       BEGIN
          STRONGCONNECT(W);
          IF LOWLINK(.W.) < LOWLINK(.V.) THEN
          BEGIN
             LOWLINK(.V.):=LOWLINK(.W.);
          END;
       END
       ELSE
       IF NUMBER(.W.) < NUMBER(.V.) THEN
       BEGIN
          K:=1;
          WHILE K <= NSTACK DO
          BEGIN
             IF STACK(.K.)=W THEN GOTO 100;
             K:=K+1;
          END;
          GOTO 200;
100:      IF LOWLINK(.W.) < LOWLINK(.V.) THEN
          BEGIN
             LOWLINK(.V.):=LOWLINK(.W.);
          END;
200:   END;
    END;
    IF LOWLINK(.V.)=NUMBER(.V.) THEN
    BEGIN
       NSCC:=NSCC+1;
       POINTER(.NSCC.):=COUNT;
       W:=STACK(.NSTACK.);
       WHILE (NUMBER(.W.) >= NUMBER(.V.)) AND (NSTACK>0) DO
       BEGIN
          IF NSTACK>1 THEN
          BEGIN
             SEQ(.COUNT.):=W;
             COUNT:=COUNT-1;
             NSTACK:=NSTACK-1;
             W:=STACK(.NSTACK.);
          END
          ELSE
          BEGIN
             SEQ(.COUNT.):=W;
             COUNT:=COUNT-1;
             NSTACK:=NSTACK-1;
          END;
       END;
    END;
END;  (*  OF STRONGCONNECT  *)
```

```
BEGIN

   (*  READ ADJACENCY STRUCTURE AND CREATE LISTS  *)

   READ(NV);
   FOR J:=1 TO NV DO
   BEGIN
      NEW(PP);
      PREDLIST(.J.):=PP;
      PREDPOINT(.J.):=PP;
   END;
   FOR J:=1 TO NV DO
   BEGIN
      NEW(PP);
      SUCCLIST(.J.):=PP;
      SUCCLIST(.J.)^.NR:=J;
      THISNODE:=SUCCLIST(.J.);
      READ(K);
      WHILE K > 0 DO
      BEGIN
         NEW(PP);
         THISNODE^.NEXT:=PP;
         THISNODE:=THISNODE^.NEXT;
         THISNODE^.NR:=K;
         READ(W);
         THISNODE^.WT:=W;
         NEW(PP);
         PREDPOINT(.K.)^.NEXT:=PP;
         PREDPOINT(.K.):=PP;
         PREDPOINT(.K.)^.NR:=J;
         READ(K);
      END;
   END;
   CLTIME(T1);

   (*  INITIALIZATION  *)

   NSTORE:=0;
   I:=0;
   NSCC:=0;
   COUNT:=NV;
   NSTACK:=0;
   FOR J:=1 TO NV DO POINTER(.J.):=0;
   POINTER(.1.):=NV;

   (*  FIRST APPLICATION OF TARJAN'S PROCEDURE  *)

   FOR J:=1 TO NV  DO
   BEGIN
      IF NUMBER(.J.)=0 THEN STRONGCONNECT(J);
   END;

   (*  CONTINUE UNTIL ALL LOOPS ARE REMOVED  *)

   WHILE NSCC < NV DO
   BEGIN

      (*  FIND FIRST SCC WITH MORE THAN ONE VERTEX  *)

      KK:=NV;
```

```
WHILE POINTER(.KK-1.) <= (POINTER(.KK.)+1) DO
BEGIN
   IF POINTER(.KK-1.) = (POINTER(.KK.)+1) THEN
   BEGIN
      L:=POINTER(.KK-1.);
      IF NOT STORED(.SEQ(.L.).) THEN

      (*  PUT SINGLE VERTICES IN FINAL SEQUENCE  *)

      BEGIN
         NSTORE:=NSTORE+1;
         FINSEQ(.NSTORE.):=SEQ(.L.);
         STORED(.SEQ(.L.).):=TRUE;
      END;
   END;
   KK:=KK-1;
END;
START:=POINTER(.KK.)+1;
IEND:=POINTER(.KK-1.);

(*  SELECT TEAR STREAM(S)  *)

FOR J:=START TO IEND DO
BEGIN
   L:=SEQ(.J.);
   RO(.L.):=0;
   TAU(.L.):=0;
   CURR(.L.):=NSCC
END;
FOR J:=START TO IEND DO
BEGIN
   L:=SEQ(.J.);
   CURRENT:=SUCCLIST(.L.);
   WHILE CURRENT^.NEXT <> NIL DO
   BEGIN
      CURRENT:=CURRENT^.NEXT;
      K:=CURRENT^.NR;

      (*  CHECK IF VERTEX IN CURRENT SCC  *)

      IF CURR(.K.)=NSCC THEN
      BEGIN
         W:=CURRENT^.WT;
         TAU(.L.):=TAU(.L.)+W;
         RO(.K.):=RO(.K.)+W;
      END;
   END;
END;

(*  CALCULATE ALFA FOR ALL VERTICES OF THIS SCC  *)

FOR J:=START TO IEND DO
BEGIN
   L:=SEQ(.J.);
   ALFA(.L.):=RO(.L.)/TAU(.L.);
END;
K:=SEQ(.START.);
MIN:=ALFA(.SEQ(.START.).);
```

```
      (*  FIND SMALLEST ALFA  *)

      FOR J:=(START+1) TO IEND DO
      BEGIN
         IF ALFA(.SEQ(.J.).) < MIN THEN
         BEGIN
            K:=SEQ(.J.);
            MIN:=ALFA(.SEQ(.J.).);
         END;
      END;

      (*  TEAR ALL INLETS TO UNIT K  *)

      FOR J:=1 TO NV DO
      BEGIN
         CURRENT:=SUCCLIST(.J.);
         PREVIOUS:=CURRENT;
         WHILE CURRENT^.NEXT <> NIL DO
         BEGIN
            CURRENT:=CURRENT^.NEXT;
            IF CURRENT^.NR = K THEN
            BEGIN
               PREVIOUS^.NEXT:=CURRENT^.NEXT;
               GOTO 300;
            END;
            PREVIOUS:=CURRENT;
         END;
300:  END;

      (*  INITIALIZE VARIABLES FOR STRONGCONNECT  *)

      NSCC:=0;
      COUNT:=NV;
      I:=0;
      FOR J:=1 TO NV DO NUMBER(.J.):=0;
      NSTACK:=0;
      FOR J:=1 TO NV DO POINTER(.J.):=0;
      POINTER(.1.):=NV;

      (*  CALL TARJAN'S PROCEDURE  *)

      FOR J:=1 TO NV DO
      BEGIN
         IF NUMBER(.J.)=0 THEN STRONGCONNECT(J);
      END;
   END;  (*  NSCC < NV  *)

   (*  PUT SINGLE VERTICES FROM FINAL CALL  *)
   (*  OF TARJAN'S PROCEDURE IN THE FINAL   *)
   (*  SEQUENCE  (FINSEQ)                   *)

   FOR J:=1 TO NV DO
   BEGIN
      L:=SEQ(.J.);
      IF NOT STORED(.L.) THEN
      BEGIN
         NSTORE:=NSTORE+1;
         FINSEQ(.NSTORE.):=L;
      END;
   END;
```

```
    (*  GO THROUGH THE PREDLISTS OF THE VERTICES  *)
    (*  IN THE FINAL SEQUENCE AND PRINT THE       *)
    (*  EDGES THAT HAVE TO BE TORN IN ORDER TO    *)
    (*  MAKE THE DIRECTED GRAPH ACYCLIC.          *)

    FOR J:=1 TO NV DO
    BEGIN
       I:=FINSEQ(.J.);
       CURRENT:=PREDLIST(.I.);
       WHILE CURRENT^.NEXT <> NIL DO
       BEGIN
          CURRENT:=CURRENT^.NEXT;
          K:=CURRENT^.NR;
          IF NOT COMP(.K.) THEN
          WRITELN('TEAR EDGE FROM VERTEX',K:4,'  TO VERTEX',I:4);
       END;
       COMP(.I.):=TRUE;
    END;
    CLTIME(T2);
    T3:=T2-T1;
    WRITELN('COMPUTATIONAL SEQUENCE:');
    WRITELN(' ');
    FOR J:=1 TO NV DO WRITE(FINSEQ(.J.):4);
    WRITELN(' '); WRITELN(' ');
    WRITELN('CPU TIME USED: ',T3:8:2,' MSEC');
END.  (*  OF TEARING PROGRAM  *)
```